Error-Free Message Transmission in the Universal Composability Framework

by

David A. Wilson

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author .....

Department of Electrical Engineering and Computer Science

May 19, 2005

Certified by .....

Ronald L. Rivest

Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Certified by .....

Chris Peikert

Thesis Co-Supervisor

Thesis Supervisor

Accepted by .....

Arthur C. Smith

Chairman, Department Committee on Graduate Students

Error-Free Message Transmission in the Universal Composability Framework

by

David A. Wilson

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis introduces models for error-prone communication channels and functionalities for error-free communication in the Universal Composability framework. Realizing these functionalities enables protocols to make use of cryptographic error-correcting schemes which are more powerful than classical codes.

First, we define new ideal functionalities $\mathcal{F}_{\text{CLOSE}}$ and $\mathcal{F}_{\text{CWT}}$ to model error-prone communication channels. Then, we define four different ideal functionalities for error-free message transmission, each providing successively stronger message delivery guarantees. Using ideal message certification, we give protocols which realize three of these functionalities for error rates up to $1/2$. Finally, we prove that the fourth functionality, which models error-free data storage, is not realizable if the error rate exceeds $1/4$.

Thesis Supervisor: Ronald L. Rivest
Title: Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Chris Peikert
Title: Thesis Co-Supervisor

Acknowledgments

First, I would like to thank Chris Peikert for his great assistance during this research. He has constantly been available to lend his knowledge, advice, and help.

Additionally, I would like to thank Ron Rivest for his assistance and wealth of knowledge, and for giving me the many opportunities I have had.

I would also like to thank Silvio Micali for giving me an energetic introduction to theoretical cryptography, and my coworkers for their assistance and friendship.

Finally, I would like to thank my family for their constant support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Any message-driven protocol—cryptographic or otherwise—requires at least a basic level of reliability of the communication between the parties, with regard to both the integrity of the messages themselves and the guarantees the channels provide related to message delivery. Classically, coding theory has dealt with the former problem and cryptography with the latter; error-correcting codes allow reliable communication even over faulty channels, and cryptography has explored the notions of authentication and secrecy of the messages that go over the channels.

Traditionally, there has been no problem in combining the results of both fields, since error-correcting codes handle the low-level details of communication itself whereas cryptography uses communication as a tool to perform specific tasks. However, recent work (such as that found in [5], [9], and [12]) has used cryptographic assumptions in creating newer, more powerful error-correcting schemes. This overlap between the two fields has subtle security implications: since the low-level mechanism for error correction now has a cryptographic component, one must carefully analyze the details of *cryptographic protocol composition* when using the communication channel to engage in cryptographic protocols.

On a binary channel (that is, a communication channel over which one only sends 0s and 1s), it is well-known that no classical error-correcting code can decode a message correctly if one-fourth of the bits become corrupted.[1] Using a cryptographically

[1]While there are technical caveats to this statement, it is essentially true in the Hamming

authenticated error-correcting scheme, however, it is possible to decode from any error rate less than one-half [12]. We wish to reap the benefits of this stronger error correction at a lower level, but also to retain the security guarantees of any protocols used at a higher level.

We therefore turn to the Universal Composability framework, a cryptographic system that can be used to guarantee the behavioral semantics of various protocols. In particular, a consequence of defining a protocol in the UC framework is *composability*—this protocol may be safely used as a subroutine of another protocol without affecting its security.

Our goal is to model error-free message transmission as a functionality in the UC framework. Using ideas from [12], we wish to design cryptographic protocols that realize these functionalities. By doing so, we allow existing cryptographic protocols to make use of recent advances in error correction without affecting their security guarantees.

(adversarial-error) model.

Chapter 2

A Brief Introduction to Error Correction

One major topic of research within the field of information theory is that of error correction. *Error-correcting codes* make message transmission more robust by compensating for *errors* that occur between the time the message is sent and the time it is received. Thus, even if a faulty transmission line, radio interference, or some other factor garbles parts of the transmission, the message itself can be recovered perfectly.

2.1 The Model of Errors and Error Correction

In the standard model of error correction, a *sender* *Send*[1] wishes to communicate with a *receiver* R. They are separated by a *channel* C. Thus, any information *Send* sends to R must first pass through C, where it may be subject to errors. That is, individual symbols within the message may be corrupted such that they will read as other symbols.

There are several popular models of the method by which C introduces errors into the sent message. In the first model, due to Shannon [15], each symbol in the sent word

[1] In most papers in the field of coding theory, the sender is denoted simply by S. However, in the field of cryptography (and specifically with regard to the universal composability framework), S is typically used to denote a *simulator*, as defined in the next section. Since this paper will refer to both of these parties extensively, they are here referred to as *Send* and *Sim* respectively.

is corrupted independently at random with a certain probability p. Alternatively, in the model due to Hamming [8], the channel is permitted to corrupt a certain fraction ϵ of the symbols in each sent message, and may do so in a "worst-case" manner.

More recently, another model for the error-introducing channel has been developed. First proposed by Lipton[10], in this model the channel introduces errors in an adversarial manner, but is limited to a "feasible" amount of computation. Thus, it is slightly weaker than the Hamming model, which uses a computationally unbounded adversarial channel. However, many cryptographic protocols similarly assume that adversarial processes are computationally bounded; we therefore feel justified in using this assumption in coding theory. The usefulness of this model over the classical Hamming model can be seen in Section 2.4.

2.2 Error-Correcting Codes and Definitions

Any specific error-correcting code operates on messages of a specific length k. A *message* is a string of k symbols from a finite *alphabet* Σ; in this paper we focus on the specific case of $\Sigma = \{0, 1\}$. The message is encoded into a *code word*, or a string of $n \geq k$ symbols from Σ; n is known as the *block length.*

The code word is then sent through a communication channel, which may *corrupt* the code word by changing some of its symbols. One can measure the amount of corruption by calculating the Hamming distance between the code word and the received word. The *Hamming distance* between two words x and y of length n, denoted $\Delta(x, y)$, is the number of symbols in which they differ. That is, $\Delta(x, y) = |\{i : x_i \neq y_i\}|$.

A *coding scheme* is a pair of efficient[2] algorithms $E : \Sigma^k \rightarrow \Sigma^n$ and $D : \Sigma^n \rightarrow \Sigma^k$. The *information rate* (or simply *rate*) R of a code is defined as k/n. A code (E, D) tolerates *error rate* δ if, for all $m \in \Sigma^k$, $\Delta(E(m), r) \leq \delta n \rightarrow D(r) = m$.

One usually wishes to decode a received word to the code word that is closest in Hamming distance. Thus, if one defines d as the minimum Hamming distance

[2]That is, running in probabilistic polynomial time.

between any two code words of the code, it is clear that one can in principle correct for $(d-1)/2$ errors; this value is known as the *unique decoding radius* of the code. Thus, with the proper error-correcting code, one can uniquely decode from an error rate of approximately $d/2n$.

2.3 Decoding Non-Uniquely: List Decoding

In some circumstances, it may be possible to decode from an error rate greater than $d/2n$, provided that one relaxes the requirements for decoding. Rather than requiring that the decoding algorithm output a unique message, one may allow it to output a short *list* of possible messages. If the error rate is less than the *list-decoding radius* ϵ, then it is guaranteed that the original message will appear in the decoded list.

More formally, a code (E, D) with block length n and alphabet Σ is $(\epsilon n, L)$-*list decodable* if, for all $r \in \Sigma^n$, there exist $\ell \leq L$ code words x of E such that $\Delta(r, x) \leq \epsilon n$. A code is *efficiently list-decodable* if there exists an efficient algorithm LD such that for all $r \in \Sigma^n$, $LD(r)$ outputs all code words x such that $\Delta(r, x) \leq \epsilon n$.

Many common codes are known to be efficiently list-decodable to a list-decoding radius far greater than the unique decoding radius. For example, the concatenated codes of Guruswami and Sudan [7] (for binary alphabets) as well as Reed-Solomon codes [6] (for large alphabets) both provide very good tradeoffs between information rate and list-decoding radius.

2.4 Cryptography and Coding Theory

In coding theory, the communication channel introduces errors that can potentially disrupt communication between two parties; indeed, for purposes of analysis the errors are often assumed to be introduced in a worst-case fashion. Thus, the channel can be seen as an *adversary* whose goal is to cause the receiver to decode incorrectly. In this model, the distinction between the standard Hamming channel and the channel described by Lipton becomes clear: the adversary of the former channel is computa-

tionally *unbounded*, whereas the adversary of the latter channel can be described by a probabilistic polynomial-time algorithm.

However, the notion of an adversary—particularly one that is computationally bounded—is very reminiscent of the field of cryptography. Indeed, work has recently been performed in using cryptographic arguments and primitives in order to prove that, despite the actions of the adversarial channel, the receiver decodes correctly with high probability. For example, the "code scrambling" method of Gopalan, Lipton, and Ding [5] uses a shared secret random string between the sender and receiver in order to randomize the code word, such that the channel can only introduce errors at random rather than adversarially chosen locations. The "private codes" described by Langberg [9] use a different approach, defining a cryptographically authenticated subset of the full code to transmit messages.

Independently, Micali, Peikert, Sudan, and Wilson considered a generalized version of authenticated codes [12]. In this paper, the authors use list decoding to generate a list of possible sent messages, then filter the list based on the authenticity of a digital signature encoded with the message. While the authors focus on the usefulness of this technique in enhancing the properties of error-correcting codes (specifically, correcting for a higher error rate at a given information rate than classical codes allow), it is also useful to note that the recovered message is authenticated. This fact is useful if one wishes to use the error-correcting codes to send messages within cryptographic protocols. Thus, the protocols for the functionalities defined later in this paper are similar to that found in [12].

Chapter 3

The Universal Composability Framework

Universal Composability (UC) is a framework developed largely by Canetti [1] for analysis of cryptographic protocols. The overall goal of the UC paradigm is to provide a modular framework for cryptography in which parties can be a part of multiple different protocols at once without negatively impacting the security of the system as a whole.

In general, it is possible for protocols to be provably secure when used in isolation, but to be insecure when used concurrently with other protocols. For example, the authentication protocol of Needham and Schroeder [13] works correctly if used alone, but is insecure when used in parallel [11].

Protocols defined according to the specifications of the UC framework, however, can be composed with other protocols without compromising their security. UC protocols are known for many common cryptographic tasks; for examples, see [1] and [3].

3.1 Real-World and Ideal-World Execution

The concept of having a "real world" and an "ideal world" is a common tool in cryptographic analysis, dating back to the work of Goldreich, Micali, and Wigderson [4].

In general, security in such a model is defined to mean that the powers possessed by an adversary during execution of a real-world protocol are no greater than in an ideal protocol in which its influence is severely limited. The UC framework formally and explicitly defines the execution process in both real and ideal systems in a way that allows composability results to be proven.

3.1.1 Ideal functionalities

The UC framework works by describing an *ideal functionality*, or a description of the properties of a "perfect" cryptographic object. For example, one could use an ideal functionality to precisely describe the security one expects from a digital signature or a commitment scheme. One can define the basic, high-level behavior that any protocol should achieve, what type of operations are and are not permitted, and exactly how much information is "leaked" to the adversary at each step of execution.

3.1.2 The Interactive Environment Z

The UC system makes use of an *environment machine* Z. Z acts as an *interactive distinguisher*; its goal is to decide whether it is interacting with a real- or ideal-world system. Z, which is initiated with 1^k (the security parameter of the system), provides each party with arbitrary input of its choice, initiates the protocol, and observes the output of each party.[1] The *view* of Z is all the information that Z obtains through this process. At the end of the protocol, Z observes the output of each other party and then generates its own output, which without loss of generality consists of a single bit.

3.1.3 Real-World Execution

During real-world execution, there exist a set of parties $P_1 \dots P_n$, an adversary A, and the environment Z. Parties $P_1 \dots P_n$ run an interactive protocol π, with inputs

[1] All parties, including adversaries and the environment, are modeled as interactive Turing machines. The specifics about the different tapes and other low-level details can be found in [1].

generated by Z. The adversary A controls all network traffic (and can therefore choose to delay or drop messages between parties), communicates arbitrary information with Z, and has the power to *corrupt* parties (which allows access to the full history of the party, as well as the ability to control that party in the future).

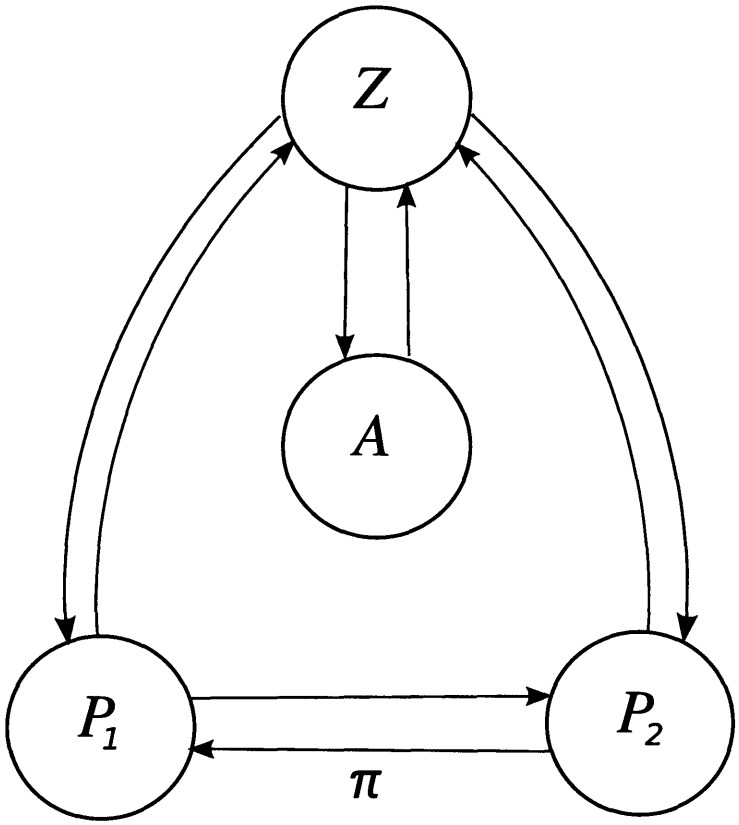

Figure 3-1: Real-world execution with two parties P_1 and P_2 running protocol π.

Let $\text{REAL}_{\pi,A,Z}$ denote the probability ensemble of the output of Z when interacting in the real-world model with parties running protocol π with adversary A, over all security parameters.

3.1.4 Ideal-World Execution

During ideal-world execution, there exist a set of *dummy parties* $P_1 \ldots P_n$, an adversary (or *simulator*) *Sim*, an ideal functionality $\mathcal{F}$, and the environment Z. As in the real-world case, Z provides input to $P_1 \ldots P_n$ and reads their output. However, instead of running a protocol, $P_1 \ldots P_n$ simply send their input to $\mathcal{F}$, and output

any message $\mathcal{F}$ sends them. The adversary *Sim* has powers similar to those of the real-world adversary; namely, it can delay or drop messages, communicate with Z, and corrupt the dummy parties.

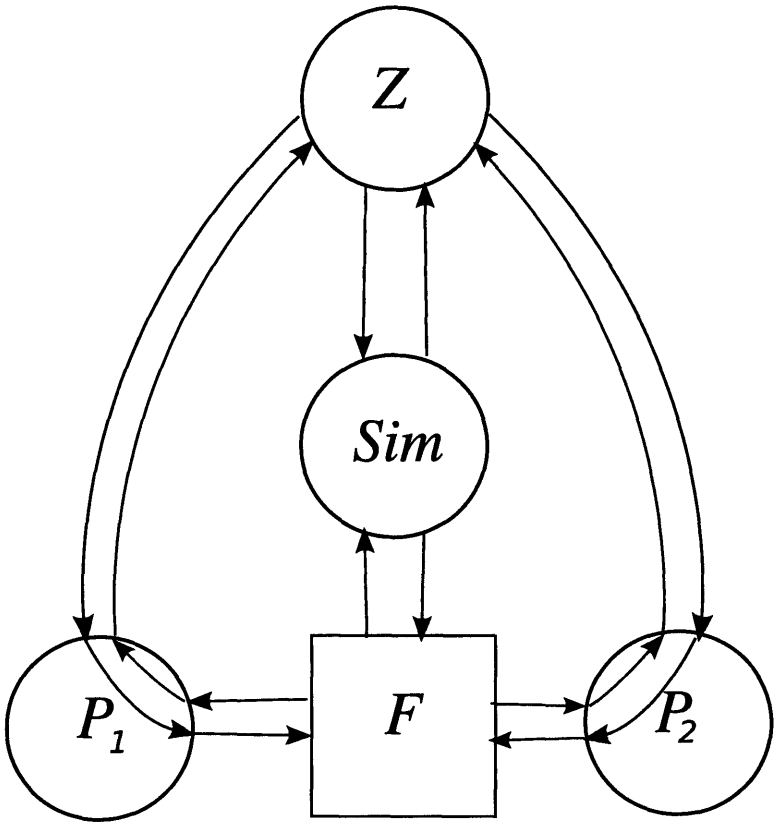

Figure 3-2: Ideal-world operation with two parties P_1 and P_2 and functionality $\mathcal{F}$.

Let $\text{IDEAL}_{\mathcal{F},Sim,Z}$ denote the probability ensemble of the output of Z when interacting in the ideal-world model with adversary *Sim* and dummy parties passing input to $\mathcal{F}$, over all security parameters.

3.1.5 Hybrid-World Execution

A *hybrid world* is similar to the real world described in Section 3.1.3, except the parties P_i and A additionally have access to one or more copies of some ideal functionalities. We motivate this "hybrid" model below.

When designing complicated protocols, it is often useful to assume access to ideal sub-functionalities. This assumption simplifies analysis and makes constructions more

modular. Additionally, in some cases a hybrid model is used to represent physical conditions, where due to the properties of the physical system one can assume a certain behavior (e.g. the functionalities defined in Section 3.2).

Let $\text{HYB}^{\mathcal{F}}_{\pi,A,Z}$ denote the probability ensemble of the output of Z when interacting in the hybrid-world model with adversary A and parties running protocol π with access to ideal functionality $\mathcal{F}$, over all security parameters.

3.1.6 Definition of Security

We say that a protocol π *securely realizes* $\mathcal{F}$ if, for every real-world adversary A there exists an ideal-world adversary Sim such that, for all environments Z, $\text{REAL}_{\pi,A,Z} \approx \text{IDEAL}_{\mathcal{F},Sim,Z}$ (that is, the two ensembles are computationally indistinguishable). If $\text{REAL}_{\pi,A,Z} \equiv \text{IDEAL}_{\mathcal{F},Sim,Z}$ (that is, if the two ensembles are identical), we say that the simulation is *perfect*.

The UC framework guarantees that protocols that securely realize ideal functionalities are *composable*. That is, they can be run simultaneously with other protocols, and in particular they can be used as sub-protocols to realize a part of a larger protocol. When analyzing large protocols, however, it is useful to assume the ideal functionality of the smaller protocol instead of re-analyzing the steps. The Universal Composability Theorem [1] proves that if protocol ρ securely realizes functionality $\mathcal{F}$, then $\text{REAL}_{\pi^\rho,A,Z} \approx \text{HYB}^{\mathcal{F}}_{\pi,A,Z}$, where π is a protocol in the $\mathcal{F}$-hybrid world and π^ρ represents a real-world protocol in which every call to $\mathcal{F}$ in π is replaced by an execution of ρ.

Thus, instead of proving indistinguishability between the real- and ideal-world views of Z, we will often prove indistinguishability between the hybrid- and ideal-world views. If these two ensembles are indistinguishable, we say that the protocol *securely realizes the functionality in the hybrid model.*

3.2 The $\mathcal{F}_{\text{AUTH}}$-, $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$-, and $\mathcal{F}_{\text{CWT}}^{\epsilon}$-Hybrid Models

The following three ideal functionalities model different types of communication channels. If a protocol runs in a hybrid model with one of these functionalities, it can run in the physical world assuming the appropriate type of communication channel exists between the parties.

The first functionality, $\mathcal{F}_{\text{AUTH}}$, models an error-free, authenticated communication channel. That is, if a message is delivered, the receiver knows the identity of the sender and that the message has not been corrupted. The $\mathcal{F}_{\text{AUTH}}$-hybrid model is sometimes called the *authenticated-links* model, and is frequently assumed in order to realize other functionalities (e.g. in [1] and [3]).

While $\mathcal{F}_{\text{AUTH}}$ has been used extensively in previous work, the next two functionalities are introduced by this paper. The second functionality, $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$, models a communication channel in which some errors can occur. Specifically, it allows the adversary to forward "corrupted" versions of sent messages to the receiver, but these corrupted words must be sufficiently close in Hamming distance to some previously sent message (as defined by parameter ϵ). Additionally, $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ only delivers messages of a specific length n; in order to deliver messages of different lengths one uses multiple copies of $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ initialized with different values n.

The third functionality, $\mathcal{F}_{\text{CWT}}^{\epsilon}$, is similar to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ in that it models an error-prone communication channel. However, $\mathcal{F}_{\text{CWT}}^{\epsilon}$ has two major differences. First, there exists an adversarially chosen, error-free *tag* associated with each message; parties may use the tag to refer to a message without knowing that message's contents. Secondly, it is *reactive*; that is, the receiver (not the adversary) initiates receiving a message. This functionality is used to model a storage medium as a communication channel, where one can store (send) a piece of data and later desire to read (receive) that specific piece of data again. (For a full exploration of the storage model, see Sections 4.3 and 8.)

Both $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ and $\mathcal{F}_{\text{CWT}}^{\epsilon}$ are defined to be *immediate*; that is, they deliver their

outputs directly to the incoming communication tapes of parties rather than allowing the adversary to asynchronously deliver the messages. Because the adversary can already control the timing of message delivery via the `corrupt` commands, non-immediacy would be redundant.

3.2.1 Functionality $\mathcal{F}_{\text{AUTH}}$

The following definition for $\mathcal{F}_{\text{AUTH}}$ is adapted from the definition found in [2].

$\mathcal{F}_{\text{AUTH}}$ proceeds as follows:

1. Upon receiving $(\mathsf{send}, \mathsf{sid}, R, m)$ from a party *Send*, send $(\mathtt{sent}, \mathsf{sid}, \mathit{Send}, R, m)$ to the adversary.

2. Upon receiving $(\mathsf{send}, \mathsf{sid}, R', m')$ from the adversary:

   (a) If *Send* is corrupted, then output $(\mathtt{sent}, \mathsf{sid}, \mathit{Send}, m')$ to R'.

   (b) Otherwise, output $(\mathtt{sent}, \mathsf{sid}, \mathit{Send}, m)$ to R, and halt.

3.2.2 Functionality $\mathcal{F}^{\epsilon}_{\text{CLOSE}}$

$\mathcal{F}^{\epsilon}_{\text{CLOSE}}$ runs with parties *Send* and R and adversary *Sim*. It is parameterized by a real number $\epsilon \in [0, 1]$ that represents the error rate it allows, and in the hybrid world is initialized with integer n. It maintains a set `sent`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \mathtt{init}, n_0)$ from *Send*:

   (a) $n \leftarrow n_0$

   (b) $\mathtt{sent} \leftarrow \{\}$

   (c) Send $(\mathsf{sid}, \mathtt{init}, n_0)$ to the adversary.

2. Upon receiving $(\mathsf{sid}, \mathsf{send}, msg)$ from *Send*:

   (a) If $msg \notin \{0,1\}^n$, do nothing. Otherwise, continue.

(b) $\texttt{sent} \leftarrow \texttt{sent} \cup \{msg\}$

(c) Send $(\mathsf{sid}, \texttt{send}, msg)$ to the adversary.

3. Upon receiving $(\mathsf{sid}, \texttt{corrupt}, x')$ from the adversary:

   (a) If there exists $x \in \texttt{sent}$ such that $\Delta(x, x') \leq \epsilon n$, then write $(\mathsf{sid}, \texttt{message}, x')$ to the incoming communication tape of R. If there does not exist such an x, or if $x' \notin \{0,1\}^n$, do nothing.

3.2.3 Functionality $\mathcal{F}_{\mathbf{CWT}}^{\epsilon}$

$\mathcal{F}_{\mathrm{CWT}}^{\epsilon}$ (which stands for "close with tag") runs with parties *Send* and R and adversary *Sim*. It is parameterized by a real number $\epsilon \in [0,1]$ that represents the error rate it allows, and in the hybrid world is initialized with integer n. It maintains two arrays, `sent[]` and `avail[]`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \texttt{init}, n_0)$ from *Send*:

   (a) $n \leftarrow n_0$

   (b) $\texttt{avail[]} \leftarrow \perp$

   (c) $\texttt{sent[]} \leftarrow \perp$

   (d) Send $(\mathsf{sid}, \texttt{init}, n_0)$ to the adversary.

2. Upon receiving $(\mathsf{sid}, \texttt{send}, \tau, msg)$ from *Send*:

   (a) If $msg \notin \{0,1\}^n$, do nothing. Otherwise, continue.

   (b) $\texttt{sent}[\tau] \leftarrow msg$

3. Upon receiving $(\mathsf{sid}, \texttt{corrupt}, \tau, x')$ from the adversary:

   (a) If $\Delta(\texttt{sent}[\tau], x') \leq \epsilon n$, then set $\texttt{avail}[\tau] \leftarrow x'$. If $\Delta(\texttt{sent}[\tau], x') > \epsilon n$, if $\texttt{sent}[\tau] = \perp$, or if $x' \notin \{0,1\}^n$, do nothing.

4. Upon receiving $(\mathsf{sid}, \texttt{receive}, \tau)$ from R:

   (a) Write $(\mathsf{sid}, \texttt{message}, \texttt{avail}[\tau])$ to the incoming communication tape of R.

3.3 The functionality $\mathcal{F}_{\text{CERT}}$

In addition to the above functionalities, which are used to model communication channels, we will also make extensive use of $\mathcal{F}_{\text{CERT}}$. $\mathcal{F}_{\text{CERT}}$ is an ideal functionality that models certification of message; that is, it binds messages to the identities of parties. The following definition for $\mathcal{F}_{\text{CERT}}$ is taken from [2], with one modification: in our version, every signature σ provided by the adversary must be of a fixed length. (This property is needed in order to ensure that the message and signature are short enough to be encoded using a particular error-correcting code.)

1. Upon receiving a value $(\texttt{sign}, \mathsf{sid}, m)$ from *Send*:

   (a) Verify that $\mathsf{sid} = (\mathit{Send}, \mathsf{sid}')$ for some sid'. If not, then ignore the request. Otherwise, continue.

   (b) Send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary.

   (c) Receive $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary. If σ does not have length equal to the security parameter, halt.

   (d) If an entry $(m, \sigma, 0)$ is recorded, then output an error message to *Send* and halt. Otherwise, continue.

   (e) Send $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to *Send*.

   (f) Record the entry $(m, \sigma, 1)$.

2. Upon receiving a value $(\mathsf{sid}, \texttt{verify}, m, \sigma)$ from some party P:

   (a) Send $(\texttt{verify}, \mathsf{sid}, m, \sigma)$ to the adversary.

   (b) Receive $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, then do the following:

      i. If $(m, \sigma, 1)$ is recorded, then set $f = 1$.

      ii. Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.

      iii. Else, if there is an entry (m, σ, f') recorded, then set $f = f'$.

iv. Else, set $f = \phi$, and record the entry (m, σ', ϕ).

(c) Send $(\texttt{verified}, \mathsf{sid}, m, f)$ to P.

This functionality provides the semantics of "certified" messages; that is, messages with an authentication tag that guarantees that they may only come from a particular sender. In practical terms, this can be thought of as providing the guarantees of digital signatures, with the addition of a "certification authority" that guarantees that a particular public key belongs to a particular party.[2].

Note: Concurrent work by Patil [14] has demonstrated a flaw in the definition of $\mathcal{F}_{\text{CERT}}$ from [2], on which the above definition is heavily based. However, it should be noted that the flaw, and subsequent fix, deal only with the interaction of the functionality and the adversary and with the secrecy properties of the signatures; the certification itself still correctly identifies a message as coming from a particular sender. Since the functionality of certification itself is all that is needed to realize the error-free communication functionalities, any future "fixed" definitions of $\mathcal{F}_{\text{CERT}}$ (such as the one proposed in [14]) can be substituted for the above.

[2]Formally, one realizes $\mathcal{F}_{\text{CERT}}$ in the $\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}}$-hybrid model, as seen in [2]

Chapter 4

Overview of Ideal Functionalities for Error-Free Communication

We define four new ideal functionalities for error-free message transmission: $\mathcal{F}_{\text{EFRR}}$, $\mathcal{F}_{\text{EFR1}}$, $\mathcal{F}_{\text{EFO}}$, and $\mathcal{F}_{\text{EFS}}$. Each is intended to provide successively higher levels of guarantees for message delivery, and models a situation found in the real world.

4.1 Motivation for Defining Functionalities for Error-Free Communication

Most protocols in the Universal Composability framework operate in the $\mathcal{F}_{\text{AUTH}}$-hybrid model. That is, they assume authenticated, error-free communication, such that a real-world adversary cannot forge messages from one party to another. The $\mathcal{F}_{\text{AUTH}}$-hybrid model is normally justified by either physical or cryptographic assumptions.

However, real-world communication is prone to errors. Radio waves experience interference, there is line noise on transmission wires, physical media experiences wear or simply degrades with time. In the $\mathcal{F}_{\text{AUTH}}$-hybrid model, corrupted messages are simply dropped; thus, in a situation where some errors are unavoidable one cannot communicate at all.

We wish to allow communication even in a realistic model where errors can occur. Thus, we use the $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$-hybrid model, where the adversary is allowed to introduce a certain number of errors into each sent message. In this model, we define protocols that enable the recipient to recover the original, uncorrupted message despite the adversarially-introduced errors. The error-free property of the data transmission is formalized by defining several ideal functionalities.

In addition to providing error-free message delivery in a weaker hybrid model, the successive functionalities also provide additional guarantees regarding ideal-world message delivery, such as prevention of adversarial reordering or duplication of messages. The semantics of these ideal functionalities, combined with a protocol that provably realizes each functionality, yield the same message-delivery properties in the real world.

In addition, due to the Universal Composition theorem [1], the protocols for the functionalities below are composable; thus, the corresponding protocols may be used to provide error correction when transmitting data for other cryptographic protocols without affecting those protocols' security.

4.2 Design Choices for the Functionalities for Error-Free Communication

All four of the new functionalities for error-free communication share some common behavior. The major common themes are highlighted and motivated below:

- Whenever a message is sent to the functionality, the functionality sends that message to the adversary. This reflects the fact that the functionalities are not concerned with secrecy but rather with error-free communication.

- The adversary may drop messages, or delay them for arbitrarily long periods of time. This not only is consistent with the adversary of the UC framework, but also reflects realistic communication channels, where network delays are

unpredictable and it is possible for messages to be dropped or to become garbled beyond hope of decoding.

- The adversary may attempt to duplicate or replay messages, corresponding to the real-world possibility of echoes or even protocols (such as TCP) that cause a message to actually be transmitted multiple times (even if theoretically one only wants one copy). Since we use the error-free communication functionalities to model a variety of different message transmission guarantees, we choose to provide these guarantees using the functionalities themselves rather than restricting the adversary.

- The adversary may *not* create new messages. Again, the error-free functionalities are meant to model a basic communication channel of the type used in coding theory, where the adversary attempts to corrupt messages but does not create new ones of its own. Once the adversary begins actively sending messages to other parties, the problem becomes an explicitly cryptographic one, and cryptographic protocols to solve this problem may be composed with protocols to realize the error-free functionality.

- The adversary may corrupt a party (either sender or receiver), learning its entire state history and controlling that party in the future. This corresponds to the fact that, for whatever reason, communicating parties may fail to follow the agreed-upon communication protocol. However, we must formally define what may happen in the communication functionality if a party fails to follow the protocol.

  In the physical world, the receiver's output is completely determined by the sender and the adversarial channel. Thus, once the sender is corrupted, the receiver's output is entirely controlled by the adversary. For simplicity, therefore, our functionalities include a `sendNow` command that enables the adversary to directly control the receiver's output, but *only* when the sender is corrupted.

Functionality	Adv. can duplicate?	Adv. can reorder?	Adv. can drop?	Who chooses message?	Realizable for $1/4 \leq \epsilon < 1/2$?
$\mathcal{F}_{\text{EFRR}}$	Yes	Yes	Yes	Adversary	Yes
$\mathcal{F}_{\text{EFR1}}$	No	Yes	Yes	Adversary	Yes
$\mathcal{F}_{\text{EFO}}$	No	No	Yes	Adversary	Yes
$\mathcal{F}_{\text{EFS}}$	(No)	(No)	Yes	Receiver	No

Table 4.1: Different properties of the error-free functionalities.

4.3 Contrasting the New Ideal Functionailities

Table 4.1 provides a comparison of the four functionalities for error-free communication presented in this paper. All four functionalities provide error-free data transmission, but each provides different semantics and additional guarantees.

- $\mathcal{F}_{\text{EFRR}}$ stands for "Error-Free Communication with Reordering and Replaying." The adversary is free to drop, reorder, or duplicate messages at will. Each message that is received, however, is guaranteed to have been sent at some point by the sender. Thus, this functionality provides guarantees similar to those one would expect of a packet-switched network. In addition, the semantics of $\mathcal{F}_{\text{EFRR}}$ are essentially the same as those of $\mathcal{F}_{\text{AUTH}}$. Effectively, $\mathcal{F}_{\text{EFRR}}$ achieves $\mathcal{F}_{\text{AUTH}}$ in the $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$-hybrid model, meaning that communication can occur normally even if the communication channels between the parties are error-prone.

- $\mathcal{F}_{\text{EFR1}}$ stands for "Error-Free Communication with Reordering and One-Time Delivery." The adversary may drop or reorder messages at will, but may not replay them. (A comparison can be made to the physical mail system: items might get lost or arrive in a different order from when they are sent, but no duplication can take place.) Thus, $\mathcal{F}_{\text{EFR1}}$ builds off of the guarantees of $\mathcal{F}_{\text{EFRR}}$ but additionally prevents the adversary from repeating messages that have already been received.

- $\mathcal{F}_{\text{EFO}}$ stands for "Error-Free Communication with Ordered Message Delivery." Messages are received in the order in which they are sent. The adversary may

drop messages, but may neither reorder nor replay them. Thus, the guarantees of this functionality can be thought of as modeling an ephemeral transmission such as a radio broadcast, where each message in turn is either received or lost forever.

- $\mathcal{F}_{\text{EFS}}$ stands for "Error-Free Storage" and provides a somewhat different model from the previous three functionalities. Rather than modeling various forms of broadcast transmission, it is intended to capture the semantics of storage on physical media. Thus, the sender "sends" (stores) a message, the adversary may corrupt it (corresponding to degradation or wear of the storage medium), and then the receiver may "receive" (access) it.

  One key difference between $\mathcal{F}_{\text{EFS}}$ and the previous functionalities is that the receiver, rather than the adversary, chooses which message to receive and when.[1] However, the question then arises as to how the receiver is to specify the message (since if he knew the message already, there would be no need to go to the functionality to receive it). Therefore, in order to model the concept of data storage in the real world, $\mathcal{F}_{\text{EFS}}$ uses the $\mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid model rather than the $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$-hybrid model. Thus, each sent message has an associated error-free "tag" that is used as an identifier. (The tag is chosen adversarially, so the sender cannot communicate via the tag.) This tag corresponds to whatever the real-world receiver would use to identify the data–for example, it could be the logo on a CD case or the sector of the hard drive where the data is stored. Regardless, in most real-world scenarios this "tag" is independent of the data it identifies; thus, the $\mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid model accurately models the behavior of real-world storage systems.

  However, as shown below, $\mathcal{F}_{\text{EFS}}$ cannot be non-trivially realized in this hybrid model if the error rate equals or exceeds $1/4$.

[1] Thus, the concept of adversarial reordering or duplication is meaningless, although the adversary still has the power to drop messages. It should also be noted that the receiver is permitted to request the same message multiple times, which could be considered a form of "duplication"; hence, we specifically refer to *adversarial* duplication as being prevented by having the receiver request messages.

It is clear that these functionalities increase in strength in the order of presentation; $\mathcal{F}_{\text{EFR1}}$ provides the guarantees of $\mathcal{F}_{\text{EFRR}}$ and additionally disallows duplication, and $\mathcal{F}_{\text{EFO}}$ provides the guarantees of $\mathcal{F}_{\text{EFR1}}$ and additionally disallows reordering. In $\mathcal{F}_{\text{EFS}}$, the adversary is prevented from choosing when the receiver decodes messages, thus preventing adversarial reordering and duplication by default.

Chapter 5

$\mathcal{F}_{\text{EFRR}}$: Error-Free Communication with Reordering and Replaying of Messages

5.1 Definition

$\mathcal{F}_{\text{EFRR}}$ is used for transmitting messages of length k. It maintains a set `pending`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from *Send*:
   - `pending` $\leftarrow \{\}$
   - Send $(\mathsf{sid}, \texttt{init}, 1^k)$ to the adversary.
2. Upon receiving $(\mathsf{sid}, \texttt{send}, msg)$ from *Send*:
   - If $msg \notin \{0,1\}^k$, do nothing. Otherwise, continue.
   - `pending` $\leftarrow$ `pending` $\cup \{msg\}$
   - Send $(\mathsf{sid}, \texttt{send}, msg)$ to *Sim*.
3. Upon receiving $(\mathsf{sid}, \texttt{allow}, msg)$ from *Sim*:

- If $msg \in$ pending, send $(\mathsf{sid}, \texttt{message}, msg)$ to R. Otherwise, do nothing.

4. Upon receiving $(\mathsf{sid}, \texttt{sendNow}, msg)$ from Sim:

   - If $Send$ is corrupted, send $(\mathsf{sid}, \texttt{message}, msg)$ to R. Otherwise, do nothing.

5.2 Protocol for $\mathcal{F}_{\text{EFRR}}$

$\mathcal{F}_{\text{EFRR}}$ can be realized by the following protocol, denoted $\pi_{\text{RR}} = (\pi_{\text{RR}}^{Send}, \pi_{\text{RR}}^{R})$, in the $\mathcal{F}_{\text{CERT}}$, $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$-hybrid model:

- The sender $Send$ runs π_{RR}^{Send}:
  - On receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from Z:
    - Generate new $\mathcal{F}_{\text{CERT}}$ session ID $\mathsf{sid}' = (Send, \mathsf{sid})$.
    - Based on an error rate of ϵ and a message length of k plus the security parameter, calculate the resulting block length n needed by the error-correcting code. Send $(\mathsf{sid}, \texttt{init}, n)$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.
  - On receiving $(\mathsf{sid}, \texttt{send}, msg)$ from Z:
    - Send $(\texttt{sign}, \mathsf{sid}', msg)$ to $\mathcal{F}_{\text{CERT}}$.
    - Receive $(\texttt{signature}, \mathsf{sid}', msg, \sigma)$ from $\mathcal{F}_{\text{CERT}}$.
    - Using an error-correcting code with encoding function E (that has efficient list-decoding algorithm LD), compute $x = E(msg \circ \sigma)$.
    - Send $(\mathsf{sid}'', \texttt{send}, x)$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.
- The receiver R runs π_{RR}^{R}:
  - On receiving $(\mathsf{sid}'', \texttt{message}, x')$ from $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$:
    - Run $LD(x')$, where LD is an efficient list-decoding algorithm for the error-correcting code used. This generates a set of possible decodings, which can be parsed as $\{m_1 \circ \sigma_1, m_2 \circ \sigma_2, \ldots m_\ell \circ \sigma_\ell\}$.

* $\texttt{verified} \leftarrow \{\}$
* For $i = 1$ to ℓ
  - Send $(\texttt{verify}, \mathsf{sid}', m_i, \sigma_i)$ to $\mathcal{F}_{\text{CERT}}$.
  - Receive $(\texttt{verified}, \mathsf{sid}', m_i, f_i)$ from $\mathcal{F}_{\text{CERT}}$.
  - If $f_i = 1$, $\texttt{verified} \leftarrow \texttt{verified} \cup \{m_i\}$
* If verified is empty, output $\perp$. Else choose $m \in \texttt{verified}$ such that m is the first element of verified in lexicographic order[1], and output m.

5.3 Proof of Security

Theorem 1. *The above protocol securely realizes $\mathcal{F}_{EFRR}$ in the $\mathcal{F}_{CERT}$, $\mathcal{F}^{\epsilon}_{CLOSE}$-hybrid world; moreover, the simulation is perfect. That is, for every hybrid-world adversary A, there exists an ideal-world adversary Sim such that* $\text{HYB}^{\mathcal{F}_{CERT}, \mathcal{F}^{\epsilon}_{CLOSE}}_{\pi_{RR}, A, Z} \equiv \text{IDEAL}_{\mathcal{F}_{EFRR}, Sim, Z}$.

Proof. Here we define a simulator *Sim* that interacts with Z, $\mathcal{F}_{\text{EFRR}}$, and "dummy" parties *Send* and R in a manner indistinguishable from parties performing the hybrid-world protocol π_{RR}.

Sim emulates the execution of the hybrid-world protocol internally, as shown in Figure 5-2. Specifically, he:

- runs an internal copy of A in a black-box manner, and sends messages back and forth between A and Z untouched.
- runs internal copies of the ideal functionalities $\mathcal{F}_{\text{CERT}}$ and $\mathcal{F}^{\epsilon}_{\text{CLOSE}}$ (which interact with the internal A).
- runs internal copies of the code π^{Send}_{RR} and π^{R}_{RR}, which communicate with the ideal functionalities that are also emulated by *Sim* in an internal "hybrid world" with adversary A.

[1] One could easily use some other method to select a message from the verified set, such as simply taking an element at random. The only requirement is that the method be easily simulatable.

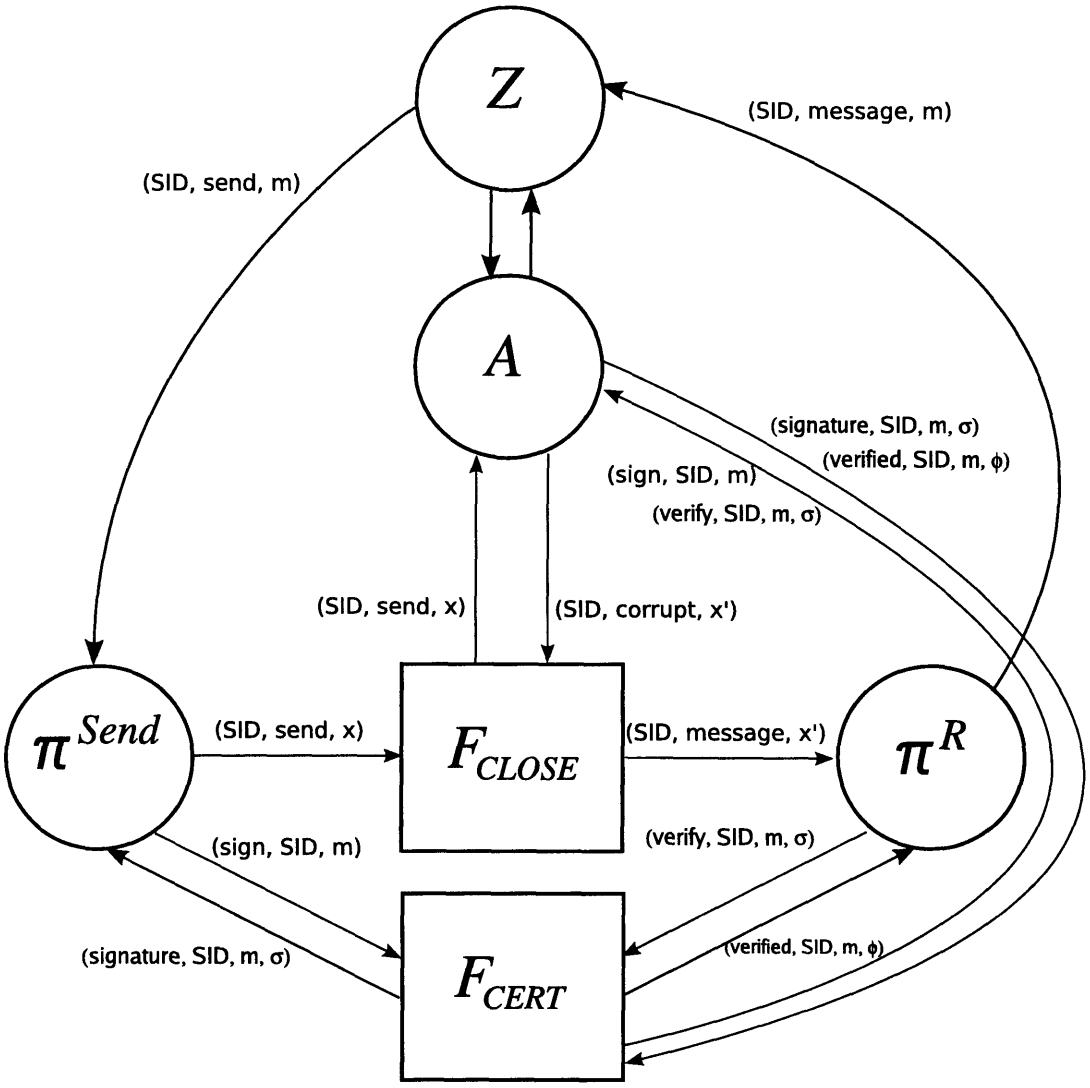

Figure 5-1: Hybrid-world operation of π_{RR}.

Sim operates in the ideal world as follows:

- When *Sim* receives a message $(\mathsf{sid}, \mathtt{init}, 1^k)$ from $\mathcal{F}_{\mathrm{EFRR}}$, he forwards that message to his internal copy of π_{RR}^{Send}.
- When *Sim* receives a message $(\mathsf{sid}, \mathtt{send}, msg)$ from $\mathcal{F}_{\mathrm{CLOSE}}^{\epsilon}$, he forwards that message to his internal copy of π_{RR}^{Send}.
- When the internal π_{RR}^{R} outputs $(\mathsf{sid}, \mathtt{message}, msg)$, *Sim* checks if *Send* has been corrupted. If so, *Sim* sends $(\mathsf{sid}, \mathtt{sendNow}, msg)$ to $\mathcal{F}_{\mathrm{EFRR}}$; otherwise, it sends $(\mathsf{sid}, \mathtt{allow}, msg)$ to $\mathcal{F}_{\mathrm{EFRR}}$.

- When the internal A requests to corrupt one of the parties ($Send$ or R), Sim corrupts that party in the ideal world and supplies the state of the corresponding internal party (π_{RR}^{Send} or π_{RR}^{R}) to the internal A.

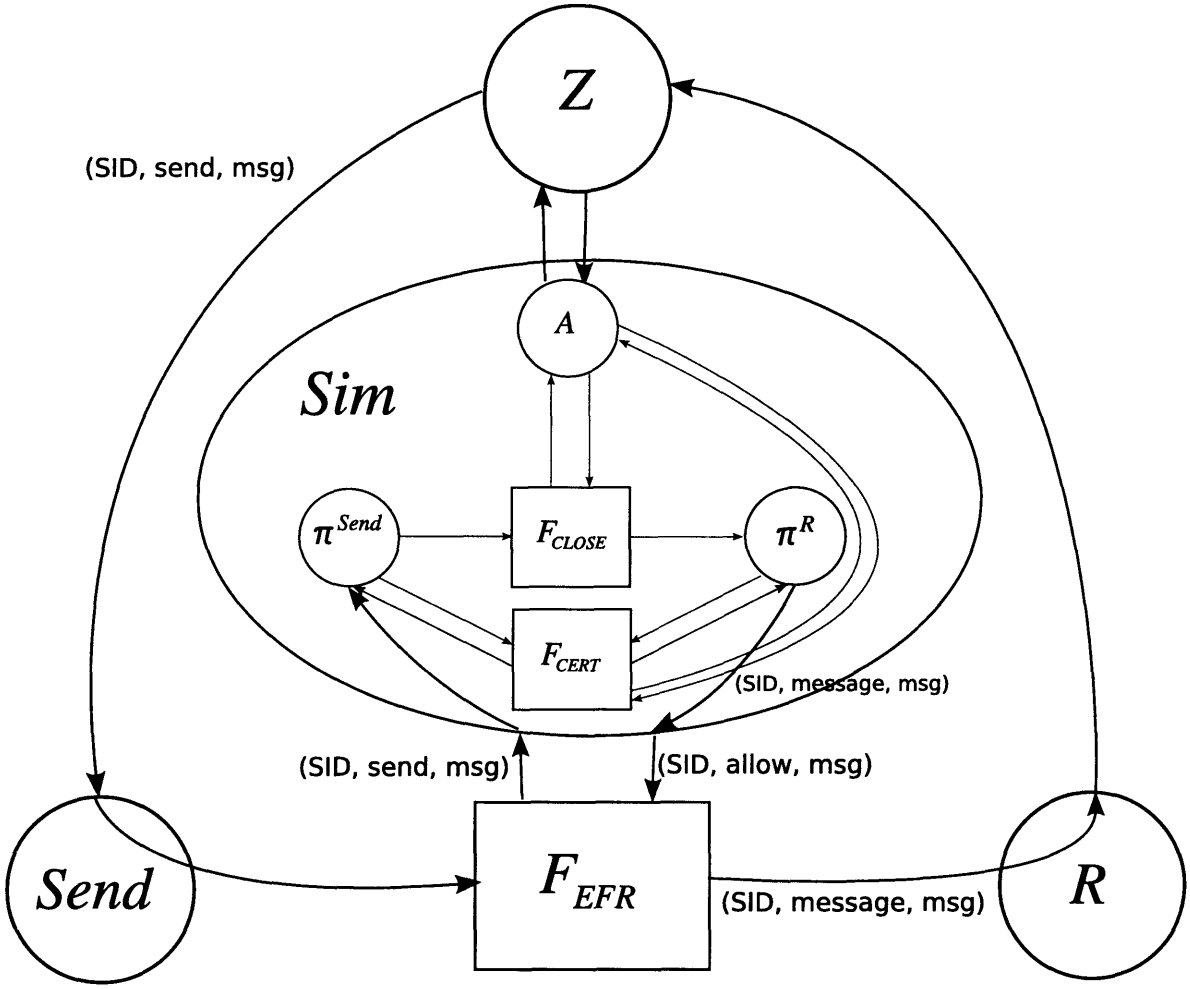

Figure 5-2: Ideal-world operation with simulator Sim.

We now consider each action Z may perform and its resulting view. Although Z may technically send any message it wants, only a few messages are defined by the protocol and functionality and will therefore result in operation. These are enumerated below:

1. Z may send a $(\mathsf{sid}, \mathtt{init}, 1^k)$ command to a party $Send$.

   In the ideal world, this command is forwarded through $\mathcal{F}_{\mathrm{EFRR}}$ to the copy of

π_{RR}^{Send} within Sim, whereas in the hybrid world $Send$ runs π_{RR}^{Send} directly on this input. In either case, π_{RR}^{Send} is run on the same input, causing a message to be sent to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$, which is then forwarded to A. Thus, Z's views of the two worlds are identical.

2. Z may send $(\mathsf{sid}, \mathsf{send}, msg)$ to $Send$.

   Hybrid world: $Send$ runs π_{RR}^{Send}, generating an encoding $E(msg \circ \sigma)$ and forwarding it to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$. $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ stores this value and forwards it to A (who may then communicate it to Z).

   Ideal world: The dummy $Send$ forwards $(\mathsf{sid}, \mathsf{send}, msg)$ to $\mathcal{F}_{\text{EFRR}}$, which in turn forwards it to Sim (after storing msg). Sim sends this instruction to his internal copy of π_{RR}^{Send}, which generates $E(msg \circ \sigma)$ and forwards it to the (simulated) functionality $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$. $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ will forward this value to the A internal to Sim, which may then communicate it to Z (since Sim faithfully forwards messages between his internal A and Z). Thus, the ideal-world view of Z is identical to the hybrid-world view, since it is based on identical (hybrid-world) sender code generating a message seen by an identical A.

3. Z may communicate arbitrary information with the adversary.

   Since Sim faithfully passes messages between his internal copy of A and Z, the view of Z generated strictly from communication with A is identical if the view of A in the hybrid world is identical to the view of the simulated A in the ideal world. However, in both cases A only sees its own communication with Z and a message from $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$ containing a code word generated by π_{RR}^{Send}. Thus, the views are identical.

4. Z may ask A to corrupt one of the parties ($Send$ or R). However, Sim is defined to reveal the state of the corresponding internal party, which is identical to the party's state in the hybrid world. Similarly, since Sim is also running A internally, it becomes a simple matter to allow A to control the execution of the internal "sender" or "receiver." Thus, the views of Z are identical in the

ideal and hybrid worlds even if a party is corrupted.

5. Z may tell A to send $(\mathsf{sid}, \mathtt{corrupt}, x')$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.

   First, note that if x' is not sufficiently close to a sent message as to be forwarded by $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$, or if it is forwarded but π_{RR}^{R} fails to output a message, then the output of R in the hybrid and ideal worlds is trivially identical. However, if π_{RR}^{R} outputs $(\mathsf{sid}, \mathtt{message}, msg)$ for some message msg, then from the definition of the protocol, $\mathcal{F}_{\text{CERT}}$ certified msg as coming from *Send*. There are two cases: if *Send* is uncorrupted, then by definition of $\mathcal{F}_{\text{CERT}}$ and π_{RR}^{Send}, *Send* had issued a $(\mathsf{sid}, \mathtt{send}, msg)$ command to $\mathcal{F}_{\text{EFRR}}$, and hence $msg \in \mathtt{pending}$. Thus, when *Sim* issues the `allow` command, $\mathcal{F}_{\text{EFRR}}$ will send $(\mathsf{sid}, \mathtt{message}, msg)$ to the dummy R.

   If *Send* is corrupted, then there is no guarantee that $msg \in \mathtt{pending}$ because $\mathcal{F}_{\text{CERT}}$ allows the adversary to determine whether or not arbitrary messages are certified. However, in this case, *Sim* issues a `sendNow` command, which causes $\mathcal{F}_{\text{EFRR}}$ to bypass the `pending` set and simply deliver the message.

   Thus, the output of R is identical in the hybrid and ideal worlds.

Thus, the view of Z is identical between hybrid and ideal worlds.

□

5.4 Discussion

In order for the receiver to be able to separate messages sent by *Send* from other decoded messages, *Send* sends a digital signature along with the message. The signature provides authentication for certain decoded messages, which is not only useful in itself, but also allows R to decode correctly. π_{RR} specifies that the message and signature are encoded as a unit; this enables candidate message-signature pairs to be decoded easily and, with a proper relation between message length and security parameter, does not result in asymptotic loss of information rate[12].

It is worth noting that the decoding algorithm used, and the related parameters such as the error rate ϵ, are independent of the above security proof. Indeed, the security of this protocol relies on the authentication provided by $\mathcal{F}_{\text{CERT}}$; the error-correction properties are used only to guarantee that R produces output in a model where some errors are unavoidable.

In addition, in the given protocol π_{RR}, both sender and receiver are stateless; they do not need to retain any information between one invocation of the protocol and the next.

Chapter 6

$\mathcal{F}_{\text{EFR1}}$: Error-Free Communication with Reordering and One-Time Delivery

6.1 Definition

$\mathcal{F}_{\text{EFR1}}$ maintains a multiset `pending`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from *Send*:
   - $\texttt{pending} \leftarrow \{\}$
   - Send $(\mathsf{sid}, \texttt{init}, 1^k)$ to the adversary.
2. Upon receiving $(\mathsf{sid}, \texttt{send}, msg)$ from *Send*:
   - If $msg \notin \{0,1\}^k$, do nothing. Otherwise, continue.
   - $\texttt{pending} \leftarrow \texttt{pending} \cup \{msg\}$
   - Send $(\mathsf{sid}, \texttt{send}, msg)$ to *Sim*.
3. Upon receiving $(\mathsf{sid}, \texttt{allow}, msg)$ from *Sim*:
   - If $msg \in \texttt{pending}$:

- pending $\leftarrow$ pending $- \{msg\}$
- Send $(\mathsf{sid}, \texttt{message}, msg)$ to R.

Otherwise, do nothing.

4. Upon receiving $(\mathsf{sid}, \texttt{repeat})$ from Sim:

   - Send $(\mathsf{sid}, \texttt{repeat})$ to R.

5. Upon receiving $(\mathsf{sid}, \texttt{sendNow}, msg)$ from Sim:

   - If $Send$ is corrupted, send $(\mathsf{sid}, \texttt{message}, msg)$ to R. Otherwise, do nothing.

The ideal adversary Sim may attempt to duplicate messages by issuing the `allow` command more than once for the same message. However, multiple `allow` commands do not result in the message being received more than once.

6.2 Protocol for $\mathcal{F}_{\text{EFR1}}$

$\mathcal{F}_{\text{EFR1}}$ can be realized by the following protocol, denoted by $\pi_{\text{R1}} = (\pi_{\text{R1}}^{Send}, \pi_{\text{R1}}^{R})$. π_{R1} operates in the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CLOSE}}^{\epsilon}$-hybrid model.

- The sender $Send$ runs π_{R1}^{Send}:
  - On receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from Z:
    * Generate a new $\mathcal{F}_{\text{CERT}}$ session ID $\mathsf{sid}' = (Send, \mathsf{sid})$.
    * Based on error rate ϵ and message length equal to k plus twice the security parameter, calculate the resulting block length n needed by the error-correcting code. Send $(\mathsf{sid}, \texttt{init}, n)$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.
  - On receiving $(\mathsf{sid}, \texttt{send}, msg)$ from Z:
    * Choose a fresh nonce r of length equal to the security parameter. Make sure that r has not been previously used as a nonce; record it so that it is not reused in the future.

* Send $(\texttt{sign}, \mathsf{sid}', msg \circ r)$ to $\mathcal{F}_{\text{CERT}}$

* Receive $(\texttt{signature}, \mathsf{sid}', msg \circ r, \sigma)$ from $\mathcal{F}_{\text{CERT}}$

* Using an error-correcting code with encoding function E (that has efficient list-decoding algorithm LD), compute $x = E(msg \circ r \circ \sigma)$.

* Send $(\mathsf{sid}'', \texttt{send}, x)$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.

- The receiver R runs π_{R1}^{R}

  R maintains a set `received` that is initialized to the empty set.

  – On receiving $(\mathsf{sid}'', \texttt{message}, x')$ from $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$:

  1. Run $LD(x')$, where LD is an efficient list-decoding algorithm for the error-correcting code used. This generates a set of possible decodings, which can be parsed as $\{m_1 \circ r_1 \circ \sigma_1, m_2 \circ r_2 \circ \sigma_2, \ldots m_\ell \circ r_\ell \circ \sigma_\ell\}$.
  2. $\texttt{verified} \leftarrow \{\}$
  3. For $i = 1$ to ℓ
     * Send $(\texttt{verify}, \mathsf{sid}', m_i \circ r_i, \sigma_i)$ to $\mathcal{F}_{\text{CERT}}$.
     * Receive $(\texttt{verified}, \mathsf{sid}', m_i \circ r_i, f_i)$ from $\mathcal{F}_{\text{CERT}}$.
     * If $f_i = 1$, $\texttt{verified} \leftarrow \texttt{verified} \cup \{m_i \circ r_i\}$
  4. If `verified` is empty, output $\perp$ and halt. Otherwise, continue.
  5. $\texttt{verified} \leftarrow \texttt{verified} - \texttt{received}$
  6. $\texttt{received} \leftarrow \texttt{received} \cup \texttt{verified}$
  7. If `verified` is nonempty:
     * For each $msg \circ r \in \texttt{verified}$, output $(\mathsf{sid}, \texttt{message}, msg)$. Output the messages in order of increasing nonce.[1]

     Otherwise, output $(\mathsf{sid}, \texttt{repeat})$.

[1] Alternatively, one could choose a different order; the only requirement is that the method used to order is easily simulatable.

6.3 Proof of Security

Theorem 2. *The above protocol securely realizes $\mathcal{F}_{EFR1}$ in the $\mathcal{F}_{CERT}, \mathcal{F}^{\epsilon}_{CLOSE}$-hybrid model; moreover, the simulation is perfect. That is, for every adversary A there exists a simulator Sim such that $\text{HYB}^{\mathcal{F}_{CERT},\mathcal{F}^{\epsilon}_{CLOSE}}_{\pi_{R1},A,Z} \equiv \text{IDEAL}_{\mathcal{F}_{EFR1},Sim,Z}$ for all environments Z.*

Proof. Again, we describe a simulator Sim that emulates the execution of the hybrid-world model, running A, $\mathcal{F}_{\text{CERT}}$, $\mathcal{F}^{\epsilon}_{\text{CLOSE}}$, π^{Send}_{R1}, and π^{R}_{R1} internally. The simulator—and analysis thereof—is very similar to that described in the proof of Theorem 1, so the redundant details are not reproduced here.

$\mathcal{F}_{\text{EFR1}}$ differs from $\mathcal{F}_{\text{EFRR}}$ in two major ways: the addition of a `repeat` command, and the fact that once a message is `allow`ed it is removed from the `pending` multiset. We cover each of these differences in turn.

First, if the simulated copy of π^{R}_{R1} outputs $(\mathsf{sid}, \texttt{repeat})$, Sim simply forwards this value to $\mathcal{F}_{\text{EFR1}}$. $\mathcal{F}_{\text{EFR1}}$ sends this value to the dummy R, which outputs it. Thus, the view of Z is the same, since in both the hybrid and ideal worlds the receiver R outputs $(\mathsf{sid}, \texttt{repeat})$.

We now show that in the hybrid world, if the sender is uncorrupted and π^{R}_{R1} outputs $(\mathsf{sid}, \texttt{message}, msg)$ for some message msg, then msg must appear in the `pending` multiset of $\mathcal{F}_{\text{EFR1}}$. By the same argument as in the proof of Theorem 1, msg has appeared in `pending` at some point in the past. Thus, we show that at least one copy of msg has not yet been removed from `pending`.

Recall that π^{R}_{R1} filters out messages with nonces it has previously seen. Thus, if π^{R}_{R1} outputs $(\mathsf{sid}, \texttt{message}, msg)$, it decoded msg concatenated with a previously-unseen nonce (and a valid signature indicating that the message indeed came from $Send$). However, since each nonce used by π^{Send}_{R1} is unique, this decoding of msg corresponds to a *specific* execution of π^{Send}_{R1} on msg, an execution which has not previously led to output by π^{R}_{R1}. Since each execution of π^{Send}_{R1} corresponds to a different `send` command to $\mathcal{F}_{\text{EFR1}}$, every `allow` command sent by Sim can be matched up to a unique `send` command to $\mathcal{F}_{\text{EFR1}}$ that had previously occurred. Thus, whenever Sim

sends $(\mathsf{sid}, \mathtt{allow}, msg)$ to $\mathcal{F}_{\text{EFR1}}$, $msg \in \mathtt{pending}$ and therefore the dummy R will output $(\mathsf{sid}, \mathtt{message}, msg)$. Thus, the views of Z in the hybrid and ideal worlds are identical.

□

6.4 Discussion

The main difference between the operation of π_{R1} and π_{RR} is the inclusion of a nonce. The nonce, signed and encoded along with the message, allows the protocol to detect and discard repeated messages. If the same message is sent by *Send* multiple times, each copy can be received once, since different copies of the message will have different nonces.

However, the inclusion of the nonce also requires that both *Send* and R keep *state*. Namely, π_{R1}^{Send} must in principle keep track of all nonces it has used (in order to avoid reusing one of them) and π_{R1}^{R} must keep track of all authentic nonces it has decoded in order to detect future duplicates. In particular, while the sender may only be required to keep a limited amount of state (if, for example, it used a counter or a timestamp for the nonce), the receiver's state is in principle unbounded since it must keep all nonces it has seen due to the possibility of message reordering.

Chapter 7

$\mathcal{F}_{\text{EFO}}$: Error-Free Communication with Ordered Message Delivery

7.1 Definition

Unlike the previous two functionalities, $\mathcal{F}_{\text{EFO}}$ is immediate. This is to ensure its semantics; we wish to guarantee that R receives messages in the order in which they are sent. If the functionality were not immediate, the adversary could reorder messages between $\mathcal{F}_{\text{EFO}}$ and the dummy R. (On the sending side, the UC framework guarantees that all incoming messages to functionalities are delivered immediately.)

$\mathcal{F}_{\text{EFO}}$ keeps a FIFO queue `pending`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from *Send*:
   - `pending` $\leftarrow \{\}$
   - Send $(\mathsf{sid}, \texttt{init}, 1^k)$ to the adversary.
2. Upon receiving $(\mathsf{sid}, \texttt{send}, msg)$ from *Send*:
   - If $msg \notin \{0,1\}^k$, do nothing. Otherwise, continue.
   - Append msg to the end of `pending`.
   - Send $(\mathsf{sid}, \texttt{send}, msg)$ to *Sim*.

3. Upon receiving (sid, `allow`) from *Sim*:

   - Remove the first element of `pending` and call it *msg*.
   - Write (sid, `message`, *msg*) to the incoming communication tape of R.

4. Upon receiving (sid, `drop`) from *Sim*:

   - Remove the first element of `pending`.

5. Upon receiving (sid, `sendNow`, *msg*) from *Sim*:

   - If *Send* is corrupted, write (sid, `message`, *msg*) to the incoming communication tape of R. Otherwise, do nothing.

Through comparison with the first two functionalities, one can see that storing the pending messages as a queue rather than a set disallows reordering (although dropping is still possible through a special instruction). By design, replaying of sent messages by the adversary is also disallowed.

7.2 Protocol for $\mathcal{F}_{\text{EFO}}$

$\mathcal{F}_{\text{EFO}}$ can be realized by the following protocol, denoted by π_{O}, in the $\mathcal{F}_{\text{CERT}}$-hybrid model:

- The sender *Send*
  - On receiving (sid, `init`, 1^k) from Z:
    - Generate new $\mathcal{F}_{\text{CERT}}$ session ID $\mathsf{sid}' = (\mathit{Send}, \mathsf{sid})$.
    - Based on error rate ϵ and a message length of k plus twice the security parameter, calculate the resulting block length n needed by the error-correcting code. Send (sid, `init`, n) to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.
    - `nonce` $\leftarrow 0$
  - On receiving (sid, `send`, *msg*) from Z:

* $\texttt{nonce} \leftarrow \texttt{nonce} + 1$
* Send $(\texttt{sign}, \mathsf{sid}', msg \circ \texttt{nonce})$ to $\mathcal{F}_{\text{CERT}}$
* Receive $(\texttt{signature}, \mathsf{sid}', msg \circ \texttt{nonce}, \sigma)$ from $\mathcal{F}_{\text{CERT}}$
* Using an error-correcting code with encoding function E (that has efficient list-decoding algorithm LD), compute $x = E(msg \circ \texttt{nonce} \circ \sigma)$.
* Send $(\mathsf{sid}'', \texttt{send}, x)$ to $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$.

- The receiver R

  R maintains a variable n_{max}, which is initially set to 0.

  - On receiving $(\mathsf{sid}'', \texttt{message}, x')$ from $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$:
    * Run $LD(x')$, where LD is an efficient list-decoding algorithm for the error-correcting code used. This generates a set of possible decodings, which can be parsed as $\{m_1 \circ n_1 \circ \sigma_1, m_2 \circ n_2 \circ \sigma_2, \ldots m_k \circ n_k \circ \sigma_k\}$.
    * $\texttt{verified} \leftarrow \{\}$
    * For $i = 1$ to k
      · Send $(\texttt{verify}, \mathsf{sid}', m_i \circ n_i, \sigma_i)$ to $\mathcal{F}_{\text{CERT}}$.
      · Receive $(\texttt{verified}, \mathsf{sid}', m_i \circ n_i, f_i)$ from $\mathcal{F}_{\text{CERT}}$.
      · If $f_i = 1$, $\texttt{verified} \leftarrow \texttt{verified} \cup \{m_i \circ n_i\}$
    * If verified is empty, if $M = \{m_i | n_i = \max_{m_j \circ n_j \in \texttt{verified}} n_j\}$ does not contain exactly one element, or if $\max_{m_j \circ n_j \in \texttt{verified}} n_j \leq n_{max}$, output nothing. Otherwise, set $n_{max} \leftarrow \max_{m_j \circ n_j \in \texttt{verified}} n_j$, and output $(\mathsf{sid}, \texttt{message}, m)$ where m is the (unique) element of M.

7.3 Proof of Security

Theorem 3. *π_O securely realizes $\mathcal{F}_{EFO}$ in the $\mathcal{F}_{CERT}, \mathcal{F}_{CLOSE}^{\epsilon}$-hybrid model; moreover, the simulation is perfect. That is, for every hybrid-world adversary A, there exists an ideal-world adversary Sim such that $\text{HYB}_{\pi_O, A, Z}^{\mathcal{F}_{CERT}, \mathcal{F}_{CLOSE}^{\epsilon}} \equiv \text{IDEAL}_{\mathcal{F}_{EFO}, Sim, Z}$ for all environments Z.*

Proof. Again, we define a simulator *Sim* that emulates execution of the hybrid-world model, running internal copies of A, $\mathcal{F}_{\text{CERT}}$, $\mathcal{F}_{\text{CLOSE}}^{\epsilon}$, π_{O}^{Send}, and π_{O}^{R}. However, while the simulators for Theorems 1 and 2 ran all of these algorithms in a black-box fashion, in this case *Sim* must examine the internal workings of π_{O}^{R} in order to give the correct commands to $\mathcal{F}_{\text{EFO}}$. Once again, many aspects of the security analysis are identical to those in the proof of Theorem 1, so we omit them here.

Sim maintains a variable n that is initially set to 0; it will periodically be updated with the value of n_{max} being maintained by π_{O}^{R}. Whenever the simulated copy of π_{O}^{R} outputs $(\mathsf{sid}, \texttt{message}, msg)$, *Sim* first checks to see if *Send* has been corrupted. If so, it outputs $(\mathsf{sid}, \texttt{sendNow}, msg)$ to $\mathcal{F}_{\text{EFO}}$ (thus "forcing" the output to the right value). Otherwise, it does the following:

1. Examines the state of π_{O}^{R} and determines the new value of n_{max}.

2. Sends $n_{max} - n - 1$ `drop` instructions to $\mathcal{F}_{\text{EFO}}$.

3. Sends $(\mathsf{sid}, \texttt{allow})$ to $\mathcal{F}_{\text{EFO}}$.

4. Sets $n \leftarrow n_{max}$.

We therefore show that the allowed message is the message decoded by the (simulated) R of π_{O}.

Since every message sent by the dummy *Send* is sent by the simulated *Send* in the same order, the nonces given to the messages by the simulated *Send* are an accurate reflection of where those messages appear in the `pending` queue of $\mathcal{F}_{\text{EFO}}$. That is, since *Send* simply increments the nonce with each sent message, the difference between two nonces corresponds exactly to the difference between the positions of the associated message in the `pending` queue.

Since the nonce is signed along with each message and the signatures are checked during the decoding process, $\mathcal{F}_{\text{CERT}}$ guarantees that if R outputs a message, that message was sent with the nonce that was decoded. Since n_{max} is the nonce of the previously decoded message (or 0 in the first activation), the difference between n_{max}

and the nonce exactly corresponds to the number of positions between the corresponding message and the head of the queue. Thus, the `drop` instructions followed by a single `allow` cause $\mathcal{F}_{\text{EFO}}$ to forward the same message as was decoded.

Since the outputs of the dummy R and the simulated R are identical, the view of Z is identical to its view of the execution of the hybrid-world protocol.

□

7.4 Discussion

Like the protocol π_{R1}, π_{O} uses a nonce to achieve additional message delivery guarantees. However, by using a specific scheme for the nonces, the protocol is able to assign more semantics: a nonce is not only a unique identifier, but contains precise information on message ordering as well.

Due to the fact that the nonces follow a set sequence, both π_{O}^{Send} and π_{O}^{R} each maintain a constant amount of state—the maximum nonce that has been sent and received, respectively.

Chapter 8

$\mathcal{F}_{\text{EFS}}$: Error-Free Storage

8.1 Definition

$\mathcal{F}_{\text{EFS}}$ keeps a local array (or hash table) `pending[]`. Its operation proceeds as follows:

1. Upon receiving $(\mathsf{sid}, \texttt{init}, 1^k)$ from *Send*:
   - `pending[]` $\leftarrow \perp$
   - Send $(\mathsf{sid}, \texttt{init}, 1^k)$ to the adversary.
2. Upon receiving $(\mathsf{sid}, \texttt{send}, \tau, msg)$ from *Send*:
   - If $msg \notin \{0,1\}^k$, do nothing. Otherwise, continue.
   - Send $(\mathsf{sid}, \texttt{send}, \tau, msg)$ to the adversary.
   - `pending`$[\tau] \leftarrow msg$
3. Upon receiving $(\mathsf{sid}, \texttt{receive}, \tau)$ from R:
   - If *Send* is not corrupted, send $(\mathsf{sid}, \texttt{message}, \tau, \texttt{pending}[\tau])$ to R. (Note that if a message with tag τ has not been sent, `pending`$[\tau] = \perp$.)
   - Otherwise, send $(\mathsf{sid}, \texttt{receive}, \tau)$ to the adversary. Wait for the adversary to reply with $(\mathsf{sid}, \texttt{receive}, \tau, msg)$. If $msg \notin \{0,1\}^k$, halt. Otherwise, send $(\mathsf{sid}, \texttt{message}, \tau, msg)$ to R.

8.2 Non-Trivial Protocols

A consequence of the syntactic definition of security in the UC framework is that protocols which "do nothing" securely realize any ideal functionality [1]. Thus, in order to meaningfully claim that a functionality is not realizable, we must restrict the class of protocols to those that are somehow "useful."

In the ideal world, let a *non-blocking* adversary be defined as one which eventually delivers every message from the functionality to the recipient. (This notion was first defined in [1].)

In the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid world, define a non-blocking adversary as one which, for each $(\mathsf{sid}, \mathtt{send}, msg)$ command that is assigned tag τ, eventually issues a $(\mathsf{sid}, \mathtt{corrupt}, \tau, x')$ command to $\mathcal{F}_{\text{CWT}}^{\epsilon}$ such that $\Delta(\mathtt{pending}[\tau], x') \leq \epsilon n$ at the time the `corrupt` command is issued. That is, the adversary eventually "lets through" every message sent to $\mathcal{F}_{\text{CWT}}^{\epsilon}$, although it is allowed to corrupt them (up to error rate ϵ). The adversary must also deliver all messages from the $\mathcal{F}_{\text{CERT}}$ functionality to their recipients. The adversary is, however, allowed to block all regular network traffic between parties.

A protocol π in the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid model *non-trivially* realizes a functionality $\mathcal{F}$ if, for every non-blocking adversary A in the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid world, there exists a non-blocking adversary Sim in the ideal world such that the usual indistinguishability property applies.

8.3 Proof of Non-Realizability

The following lemma is a well-known result in coding theory and will be useful in the proof of the main theorem:

Lemma 1. ***Plotkin Bound.*** *For any $2n+1$ strings $x_1, \ldots, x_{2n+1} \in \{0,1\}^n$, there exist i, j such that $i \neq j$ and $\Delta(x_i, x_j) < n/2$.*

Theorem 4. *$\mathcal{F}_{EFS}$ cannot be non-trivially realized in the $\mathcal{F}_{CERT}, \mathcal{F}_{CWT}^{\epsilon}$-hybrid world for any $\epsilon \geq 1/4$.*

Proof. Assume there exists some protocol π that realizes $\mathcal{F}_{\text{EFS}}$ in the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid world with $\epsilon \geq 1/4$. We describe an environment Z that can distinguish between parties *Send* and R running π and interaction with a simulator and $\mathcal{F}_{\text{EFS}}$ in the ideal world.

We first observe that in π, every `send` command must result in the sender immediately making at least one call to $\mathcal{F}_{\text{CWT}}^{\epsilon}$. Consider a Z which issues repeated `send` commands with random messages and random tags, and a non-blocking adversary which immediately and trivially corrupts every message sent to $\mathcal{F}_{\text{CWT}}^{\epsilon}$. If at any point one of these requests (say, having tag τ) does not result in an immediate call to $\mathcal{F}_{\text{CWT}}^{\epsilon}$, Z could ask A to trivially corrupt the message with tag τ and then ask R to receive that message, and then halt. In the ideal world, since *Send* is a dummy party, the receiver outputs the proper message. However, in the hybrid world, R has no information about the message; Z can therefore distinguish the two worlds based on the output of R. This contradicts the assumption that π non-trivially realizes $\mathcal{F}_{\text{EFS}}$.

We now describe a Z that can distinguish in the general case. After initiating the system, Z issues repeated `send` commands with random messages and random tags. Additionally, Z instructs A to never issue a `corrupt` command to $\mathcal{F}_{\text{CWT}}^{\epsilon}$ for any tag not generated by Z.[1] Z continues until two distinct messages, m_i and m_j, result in strings x_i and x_j being sent to $\mathcal{F}_{\text{CWT}}^{\epsilon}$ that are within Hamming distance $2\epsilon n$ of each other, where n is the length of the encoded messages. Since $2\epsilon n \geq n/2$, the Plotkin bound implies that Z can find two such strings m_i and m_j within $2n+1$ `send` commands. Denote the tags of m_i and m_j by τ_i and τ_j respectively.

Define $M(x_i, x_j)$ to be some canonical value such that $\Delta(x_i, M(x_i, x_j)) \leq \epsilon$ and $\Delta(x_j, M(x_i, x_j)) \leq \epsilon$). Z then chooses τ to be equal to either τ_i or τ_j at random with equal probability and instructs the adversary to send $(\mathsf{sid}, \texttt{corrupt}, \tau, M(x_i, x_j))$ to $\mathcal{F}_{\text{CWT}}^{\epsilon}$. Z then instructs R to receive the message with tag τ.

If the environment is interacting with *Sim* and $\mathcal{F}_{\text{EFS}}$ in the ideal world, the ideal functionality will, by definition, always return the message correctly corresponding to τ when the dummy R issues the `receive` command. If, however, the environment

[1] This is necessary to prevent a protocol from embedding information in the tags.

is interacting with parties in the hybrid world, R may request several tags from $\mathcal{F}_{\text{CWT}}^{\epsilon}$, but since the tags are assigned randomly, τ is the only one which will result in any useful information. R must then choose which message to return. Since τ is independent of the $M(x_i, x_j)$ value R receives, the correct output is m_i or m_j with equal likelihood. Thus, any method R uses to select which method to output must give the incorrect message with probability at least $1/2$.

Thus, with significant probability, Z can distinguish between interaction with $\mathcal{F}_{\text{EFS}}$ and interaction with any simulator Sim. Therefore, $\mathcal{F}_{\text{EFS}}$ cannot be realized by any protocol in the $\mathcal{F}_{\text{CERT}}, \mathcal{F}_{\text{CWT}}^{\epsilon}$-hybrid model for $\epsilon \geq 1/4$.

□

8.4 Discussion

Unlike the previous functionalities for error-free communication, $\mathcal{F}_{\text{EFS}}$ is not realizable under higher error rates than those provided by classical coding theory. The reason for this is that the receiver, not the adversary, chooses the message to be received. Under the transmission-like functionalities, the adversary could choose to make the receiver decode to any of a number of messages, but since each of these messages was indeed sent by the sender, the behavior was considered correct.

In the storage model, however, the receiver wants a *specific* message sent by the sender. The adversary is thus able to make the receiver decode to a *different* message (or at least to cause ambiguity between several messages)–although all the messages the receiver is considering were indeed validly sent by *Send*, the receiver does not know which message is the one he actually wants at the moment. Thus, the stricter requirements of the receiver in the storage model cause the functionality to be unrealizable.

Bibliography

[1] Ran Canetti. A unified framework for analyzing security of protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(16), 2001.

[2] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `http://eprint.iacr.org/`.

[3] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[4] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing,* New York, pages 218–229. ACM Press, 1987.

[5] Parikshit Gopalan, Richard J. Lipton, and Yan Zong Ding. Error correction against computationally bounded adversaries. Manuscript, October 2004.

[6] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *FOCS*, pages 28–39, 1998.

[7] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 181–190, New York, NY, USA, 2000. ACM Press.

[8] Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.

[9] Michael Langberg. Private codes or succinct random codes that are (almost) perfect. In *FOCS*, pages 325–334. IEEE Computer Society, 2004.

[10] Richard J. Lipton. A new approach to information theory. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, pages 699–708. Springer-Verlag, 1994.

[11] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[12] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction against computationally bounded noise. In *Proceedings of the Second Theory of Cryptography Conference*, pages 1–16. Springer-Verlag, 2005.

[13] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[14] Akshay Patil. *On Symbolic Analysis of Cryptographic Protocols.* Master of engineering thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2005.

[15] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.